UNIX 与 Windows 平台间通信的实现
Realization of File Transfer from Point to Point Between UNIX and Windows

陈 攀
Chen Pan
(广西艺术学院,广西南宁 530022)
(Guangxi Arts College,Nanning,Guangxi,530022,China)

摘要:通过套接字(Socket)编程,在网络中客户端采用 Java 语言,服务端采用 C 语言,成功地实现不同平台下点对点的文件传输,实现网络上任何 2 台计算机间的文件共享。

关键词:文件传输 套接字编程 UNIX Windows
中图法分类号:TP393.39

Abstract: The point to point file transfer between different platforms in a network is realized using socket programme. The Java language is used in the client of the network and the C language is used in the server of the network. The documant sharing in any two computers in the network is achieved.

Key words: file transfer, Socket programme, UNIX, Windows

一直以来,UNIX 以其优秀的安全性能占据着众多单位的核心网络,但随着 Windows 的不断发展,因其操作界面友好,使得大多数单位在使用中出现了两大操作系统共存的情况,这就提出了如何在这两大操作系统进行实时通信的问题。本文通过套接字(Socket)编程,成功地实现了 UNIX 与 Windows 平台下实现点对点的文件传输,较好地解决了这个问题。

1 进程间的通信原理

1.1 Socket 编程

在基于 TCP/IP 协议的网络中,Socket 是网络通信的基本操作单元。它提供了不同主机间进程双向通信的端口,这些进程在通信前各自建立 1 个 Socket,并通过对 Socket 的读/写操作以实现网络通信。基于 TCP/IP 协议的 Socket 编程是一种典型的会话编程方式,它可适用于客户/服务通信方式,还能适用于点对点通信方式。利用 Socket 实现文件的传输有别于 FTP 协议实现的文件传输。利用 FTP 协议实现文件传输需要一个专门的服务器和客户端,并对其做较复杂的设置,且网络中的计算机之间不能随意进行通信,显然这种通信方式存在着很大的限制。利用 Socket 点对点的文件传输只要求在网络中任意 2 台计算机中随意指定服务器和客户机,就可以达到网络上任何 2 台计算机的文件共享。

2004-05-28 收稿。

1.2 程序设计思想

TCP/IP 协议允许创建和维护与远程计算机的连接,使其彼此可以进行数据传输。利用 TCP/IP 协议通讯必须分别建立客户应用程序和服务器应用程序。在利用 Socket 编程时,一般工作模式是由客户进程向服务器进程发出请求,服务器进程执行被请求的任务并将结果返回给客户进程。具体步骤为[1]:

(1)服务进程首先创建一个套接口,使用 Socket()调用;然后,将该套接口与本机的 IP 地址和某一空闲端口相关联,使用 Bind()调用;这时,服务端就可以用 Listen()调用来侦听来自客户程序的数据;套接口一旦处于听模式,服务进程将可以接收一个连接,并允许传递数据,使用 Accept()调用来完成;最后使用 Read()调用来读入数据,同时,还可以用 Write()调用来向发送进程写回一些数据,如确认信息或回显信息。

(2)客户进程也是首先创建一个套接口,使用 Socket()调用;然后,客户进程就使用 Connect()调用试图连接一个服务;连接成功之后,就可以利用 Write()调用向服务器发送数据,同时,还可以使用 Read()调用读取服务器写回的数据。需要注意的

是:在创建客户应用程序时,必须知道服务器计算机名或其IP地址、及服务器计算机进行侦听的端口,然后调用Connect方法。

(3)服务端进程或客户端进程均可调用Close()来撤销Socket并中断连接。

目前的网络一般都支持TCP/IP协议,UNIX和Windows也都提供相应的编程接口,用户可以随心所欲地编制出合乎自己要求的通信程序。现行大多数的应用程序间的通信采取的就是这种方式。

2 通信环境的建立

进程的服务端为SCO UNIX,采用C语言编程。客户端选用Windows 9x编程,环境为Micosoft的Visual J++。UNIX主机的IP地址为:172.168.17.1,Windows机的IP地址为:172.168.17.2。通信环境如图1所示。

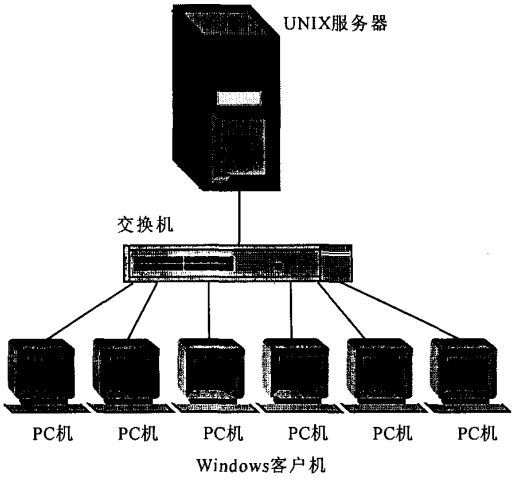

图1 实现通信的通信环境

3 程序的实现

3.1 服务端的程序[2,3]

用C语言实现。

```
/*server.c 运行于SCO UNIX  Systemv */
#include<stdio.h>
#include<siganal.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<sys/time.h>
#include<netinet/in.h>
#include<netdb.h>
struct sockaddr_in sin={AF_INET};
char liu[ ]="hello,let's begin.";
char masjh_str[60];
main (argc,argv)
char **argv;
{
   char buf [2048];
   int i,seq,nomelen,newsock,sock;
   if((sock=socket (AF_INET,SOCK_
STREAM,IPPROTO_TCP))<0)
   {
    perror ("socket");
    exit(1);
   }
   Sin.sin_port=htons(1032);/* 指定通讯
时服务器采用的端口号 */
   if(bind (sock,&sin,sizeof (sin))<0)
   {
    perro ("bind");
    exit(2);
   }
   Namelen=sizeof (sin);
   if (getsockname (sock,&sin,&namelen)<0)
   {
    perror("getsockname");
    exit(3);
   }
  printf"Server bound to port %u 0x% \n",ntohs
(sin.sin_por),sin.sin_port);
   if (fork())exit(0);
   if(listen (sock,10)<0)
   {
     perror("listen");
     exit(4);
   }
  namelen=sizeof(sin);
      if ((newsock = accept (sock, &sin,
&namelen))<0)
   {
      perror("accept");
      exit(5);
   }
  seq=0;/* 接收从windows端的消息 */
  while (read(newsock,masjh_str,1)>0)
   {
   printf"%c", masjh_str[0]);
   i++;
   putchar(7);
   if masjh_str[0]= ='$')break;
   }
```

```
    printf("receive ok! \n");
    /* 开始循环,将字符串和数字不断发往客户端 */
    for(;;)
      {
      sprintf (masjh _ str,"=unix server is beginning... =");
      write (newsock,masjh _ str,strlen(masjh _ str));
      seq++;
      if (seq>5000)seq=0;
      }
    }
```

3.2 客户端的程序

用 Microsoft 的 Visual J++ 实现。

```
//完成自 UNIX 主机发命令,并从 UNIX 取回结果
inport java.net. * ;
inport java.io. * ;
class NtHost{
    public stactic void main (String args [])
throws Exception{ int C,i;
ByteArrayOutputStream bos=new ByteArrayOutputStream();
            FileOutputStream fos = new FileOutputStream("3.txt");
    //Send command string
     byte liu3[]={(byte ) 0x41,(byte)0xc1,(byte)0xf5,(byte)'$'};
    byte liu[]=newbyte[1];
     Socket s=new Socket("172.168.17.1",1032);// 指定服务器 ip 地址和通讯端口
    InputStream in=s.getInputStream();
    OutputStteom out=s.getOutputStreom();
    out.write(liu3);
    out.flush();
    i=0;
System.out.print("Send Ok!");
while ((in.read(liu))! =-1&& i<100)
    {
    fos.write(liu);
    i++;
    }
    s.close ( );
    fos.close ( );
    System.Out.print("receive 0k!");
    }
}
```

这里要注意的是,Visual J++环境下在生成一个工作间和项目后,应在项目菜单的 Setttng…,然后在右窗口的 Debug/Category 下选择 General,类名选择 NtHost,在 Debug/Execute project under 下,应选择 Stand-alone interpreter(appliations only)。

以上 2 个程序在建立 Socket 连结后,服务端不断从套接字中读取字符流,直到遇到'$'为止,然后就不断地向 Windows 的客户端进程写入字符,而客户端不断截获由服务端发来的字符串,并写入本地的 3.txt 文件。

4 结束语

由于利用 TCP/IP 协议通讯时必须在客户端应用程序中绑定服务器的 IP 地址和与服务器计算机进行通讯时使用的端口来查找服务器及其通讯端口,所以在实际使用的网络环境中,一旦服务器的 IP 地址发生改变或者由于防火墙对通讯端口作出限制,服务器端应用程序原来绑定的通讯端口号和客户端程序原来绑定的 IP 地址和通讯端口号也必须作出相应的修改,否则作将对系统使用造成影响。

由于 Java 所定义的字符采用 16 位(bit)的拉丁 Unicode 字符集,其字符的高位为零,因此对字符的处理若简单地采用 C 语言中的:(char)c=in.read()或 out.write(c)处理高位为 1 的字符串(例:汉字),则会得到错误的结果。所以必须进行特殊的处理。此外,因 Java 提供了字节记(Byte Streams)的基本数据处理单元,而字节流的最小单元为字节(byte),这样就将 UNIX 发来的字符流转化成字节流来处理,从而成功达到同 UNIX C 进程通信的目的。

参考文献:

1 毛德操,胡希明. Linux 内核源代码情景分析(上、下). 杭州:浙江大学出版社,2001. 842.

2 Richard Stevens W. UNIX 环境高级编程. 尤晋元,等译. 北京:机械工业出版社,2000. 540.

3 Richard Stevens W. UNIX 网络编程. 杨继张译. 北京:清华大学出版社,2002. 580.

(责任编辑:邓大玉)